On the Development of Hardware Description Languages

Borrione, Dominique

Veröffentlicht in:
Jahrbuch 2001 der Braunschweigischen Wissenschaftlichen Gesellschaft, S.77-94

J. Cramer Verlag, Braunschweig

DOMINIQUE BORRIONE, Grenoble, France

On the Development of Hardware Description Languages

I. Introduction and Essential Concepts

Hardware description languages (HDL's) were invented in the late sixties as a means to describe, document, and communicate the description of digital system designs. And initially there was not necessarily an implementation. For example, the PMS and ISP languages were invented by Bell and Newell as the support for the description of computer architectures, in their famous book [BN71].

Soon thereafter, compilers were written, and hardware description languages were used as input to automatic design software:

- simulators have made it possible to consider HDL models as virtual prototypes for the verification and the performance evaluation of designs,
- hardware descriptions written in HDL's were used for macro-code generation, automatic synthesis (initially the selection of IC's, placement and routing on-board), test pattern generation and design-rule checking

The first hardware description languages inherited notions and syntax from the programming languages of the sixties. The way of writing identifiers and declarations, the distinction between formal and actual parameters of functions and procedures came from PL1 and Algol. Operators taking as arguments vectors and arrays came from APL. Yet, some significant differences distinguish hardware description languages from programming languages, among which some of the most obvious ones are:

- successive statements are sequential in programming, they are concurrent in HDL's,
- wires and registers are distinct value holders in HDL's while there is a single notion of variable in programming,
- the reference to past values of carriers is a systematic capability of many HDL's, this notion is unknown in programming,
- the notion of the rising edge and falling edge of a signal which means an event with no duration has been invented in HDL's,
- HDL's often provide an explicit sequencing control model: automata, Petri nets, …

Figure 1 shows the example of a combinational comparator written in VHDL. The circuit takes two 32-bit inputs A and B, and two 1-bit outputs AGRB and ALTB, which provide the comparison result according to the following conventions:

AGRB = 1 if a > b,	AGRB= 0 and ALTB = 0 if a = b,
ALTB = 1 if a < b,	AGRB = 1 and ALTB = 1 is impossible.

---

* Invited presentation at the Colloquium of the Braunschweigische Wissenschaftliche Gesellschaft in the honor of Professor Robert Piloty, Braunschweig, May 18, 2001

Example

```
entity COMPARE is
      port ( A, B: in Bit_vector ( 0 to 31 );
           AGRB, ALTB: out Bit );
end COMPARE ;

architecture SPEC of COMPARE is
begin
      AGRB <= '1'  when A > B  else '0' ;
      ALTB <= '1'  when  A < B  else '0' ;
end SPEC ;
```

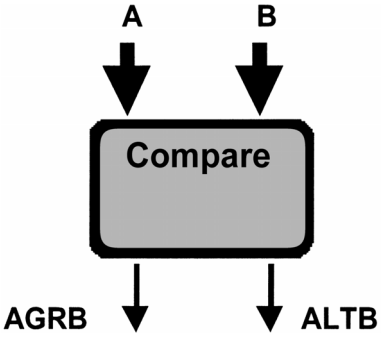


Figure 1

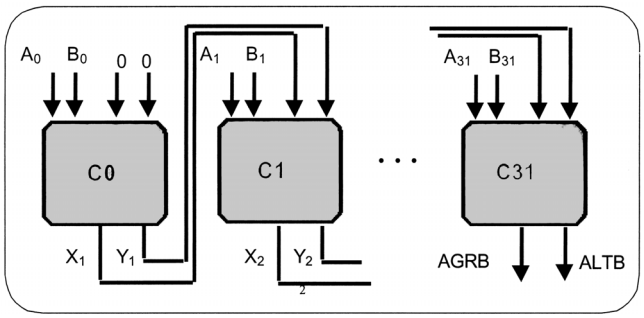
The VHDL description of the 32-bit comparator is divided in two parts : (1) the first part, called e n t i t y , describes the input-output interface of the circuit; (2) the second part, called a r c h i t e c t u r e , is the description of the inside of the box. A r c h i t e c t u r e and e n t i t y in pair constitute a component. Thus, the architecture called S P E C is one possible implementation for the comparator, whose input output interface is given by entity COMPARE. The architecture itself is described in dataflow style: each output (AGRB and ALTB) is concurrently assigned with a conditional expression which gives its value, according to the comparison of the bit-vectors A and B.

Figure 2 gives a behavioral description of the comparator, again in VHDL. Within a process, an algorithm describes the behavior specified above. It is written in a kind of programming language, in the sense that inside the process the statements are sequential. However, the figure displays a characteristic of hardware description: the process is triggered by each change of value of either A or B and executed. In other words, the process is sensitive to any event on the signals A and B considered input to the process.

```
architecture ALGORITHMIC of COMPARE is
begin
      Comploop: process (A, B)
                variable L, G: Bit ;
      begin
        L:= '0' ; G := '0' ;
        for i in 0 to 31 loop
                if A(i) and not B(i) then G := '1' ; exit ; end if ;
                if not A(i) and B(i) then L := '1' ; exit ; end if ;
        end loop ;
        AGRB <= G ;  ALTB <= L ;
      end process ;
end ALGORITHMIC ;
```

Figure 2

(a)



(b)

```
architecture IMPL of COMPARE is
        component COMP_CELL is
                port ( A, B, X, Y: in Bit ; X-CC, Y-CC: out Bit ) ;
        end component ;
        signal X, Y: Bit_vector ( 0 to 32 ) ;
begin
        CELL_CONNECT: for I in 0 to 31 generate
                C: COMP_CELL port map (A(I), B(I), X(I), Y(I), X(I+1), Y(I+1)) ;
        end generate ;
        X(0) <= '0' ; Y(0) <= '0' ;
        AGRB <= X(32) ; ALTB <= Y(32) ;
end IMPL ;
```

Figure 3

A possible implementation of the comparator is shown on Figure 3. It is given as a structural decomposition in terms of 32 1-bit comparison cells, the result of each cell being propagated as input to the next one. This structural architecture is described in VHDL by the text of Figure 3-b where the cell is declared as a component with four 1-bit inputs and two 1-bit outputs, and the repetitive interconnection of cells is shown in the generate loop which is in fact a macro-generation statement.

Finally the basic cell can further be composed in terms of a gate network using the usual logic gates as shown on Figure 4. As a result, the overall architecture is a two-level hierarchy with gates as most elementary components.

Figures 1 to 4, exhibit some of the basic concepts of hardware description languages:

- The notion of an interface which is shared between the component and its environment, and constitutes the only visibility and communication between the inside of the component its environment.
- The notion of multiple descriptions of the inside of the component with respect to its interface.
- The distinction between structural, dataflow and behavioral description.

All these concepts have been systematically defined and identified in the syntax of the VHDL standard. However, it took over 20 years to make these concepts so evident.

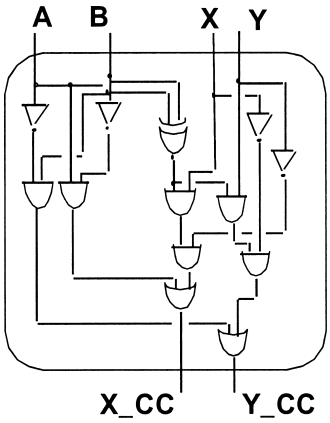


Figure 4

II. A brief history of HDL's

The historical development of hardware description languages is now recalled. The first period between 1966 and 1973 saw the discovery of the essential concepts for isolated design levels: the gate network level, the register transfer level and the micro-program level (then called the procedure level). The precursors in the USA have been LOTIS, CDL, DDL, AHPL, LOGAL, PMS, ISP, SDL, ALERT, LALSD [CHDL74, CDD92]; and in Europe: RTS1[Pi69], CASSANDRE[ME73]. The semantics of these hardware description languages were given in words, in terms of equivalent circuits or equivalent gates, and the timing aspects using chronograms.

From 1973 to 1978, primitives common to more than one design level were identified. The temporal and memory properties of various kinds of carriers were distinguished from their value types. The semantics of the languages were given in terms of their simulation. Among the languages that displayed new features : HILO[FMS75] was based on a large number of primitive functions, DIGITEST2[Ra75] had a control graph, CSL had a large number of primitive carrier types, ERES [BKS74] and RTS3[Pi75] had both a functional and a network structure, LASCAR[Bo75] introduced the abstraction from the bit data type.

During the period 1976-1985, languages became multi-level[Ha93, MN93, Ra93, MMR98]; it became evident that a single language could not alone serve all designers and all purposes, and extension mechanisms were added to the language primitives. It became obvious as well that a standardization of the ways of writing and understanding hardware descriptions was necessary. The first standardization effort was started in 1973, producing the CONLAN report [PBB83], leading the way to what became VHDL.

Between 1985 and 1995, two languages, VHDL[Ie87,Ie93] and Verilog[Ie95], were approved as IEEE standards. Formal semantics appeared for hardware description languages.

These languages became bigger and bigger with the development of mixed mode analog and digital languages and simulators[BL87]. Object orientation was added to the previous primitives and there was a come-back of graphical for well-known structural and control concepts.

The current period, started in 1993, has seen the widespread use of the two standards, with analog extensions (VHDL-AMS, Verilog-A). Both languages have become so large and so simulation-oriented that many primitive statements have no hardware semantics. Working groups were formed to define synthesizable sub-sets and synthesis-oriented libraries to provide standardized restrictions acceptable to automatic synthesis tools[Ie99]. On the other side, these languages now provide a link to other paradigms: software modules for hardware-software co-design, externally defined environment behaviors for system-on-chip design. Fast simulation and rapid prototyping are the key requirements for many current designers.

III. From a Babel Tower to a Consensus Language

When the first International Workshop on Computer Description Languages was organized at Rutgers, N.J., USA in 1973, over twenty HDL's were in existence. Each year thereafter, several more have been published, most of which added little conceptual progress to the state of the art. Jack Lipovski, in a famous article [Li77], compared the HDL community to a Tower of Babel, saying: *„Languages for describing hardware have existed since von* Neumann described his computer architecture... One of the key problems in the proliferation of rather ineffectual hardware languages has been the success of simulation as a design or analysis tool. Anyone who writes a simulator feels entitled to design his own language... *Everyone is talking a different language and nobody is listening...* "

He created and chaired the "Conference on Digital Hardware Languages", a committee of 60 scientists who aimed to develop a common syntax and set of conventions for the various levels and description tasks, resulting in a "CONsensus LANguage" (Conlan) to express the essential concepts needed for describing digital designs. The original intention was to divide this unified notation into sublanguages, to keep it learnable.

During the first two years of activity between 1973 and 1975, the committee of 60 regularly exchanged memos and ballots by post, to make proposals and vote on the scope, objectives and constructs that were to be included in the consensus language. After two years it was obvious that a more restricted group was needed to finalize the effective definition of the language.

The Conlan working group was formed at the end of 1975, consisting of Robert Piloty (chairman), Yaohan Chu soon replaced by Mario Barbacci, Dominique Borrione, Donald Dietmeyer, Fredrick Hill and Patrick Skelly.

In the absence of international networks, the working group could only communicate by post and during one or two face to face meetings a year. It took the working group until 1981 to arrive at a format and a semantic definition that could be published. The Conlan report was printed early 1983[PBB83].

All the members of the working group were the author of one previous language or had been closely related to the implementation of one such language. It was not easy to find, even among six persons, a common agreement. The real step forward was obtained when Professor Robert Piloty made the proposal to forget about all the constructs and operators voted by the 60 members of the conference, and also forget about the individual languages developed by the working group members. The group had to start from a scratch. Robert Piloty gave the impulse at the third meeting, where he brought a memo proposing: (1) a minimum set of basic principles and (2) a constructive method to define the language from a set of primitive notions which are in essence mathematical set theory. It then took several years to the group to finalize a clean construction, taking this initial idea as starting point.

The innovative idea of the Conlan working group was to define not just a hardware description language, but rather a formal method allowing to define a family of related yet simple hardware description languages. Thus Conlan is having not one but two populations of users:

- the language designers have access to all the concepts and notations defined in Conlan,
- the hardware designers have access to only those primitives which are useful for describing the structure and the behavior of digital systems.

In the following, the courrier font is used for Conlan keywords and predefined identifiers; those ending with character "@" are reserved for language definition.

IV. Base Conlan

The common core language from which all the other languages are defined is called Base Conlan (BCL). The main language primitives to describe hardware are now briefly discussed :

- The DESCRIPTION defines the model of a circuit, or a part thereof; one or more instances of a DESCRIPTION can be embedded in an enclosing DESCRIPTION.
- Two constructs are invoked to model behavior: the FUNCTION which returns a value, and the ACTIVITY which modifies one or more parameters (analogous to a procedure, but built with concurrent statements). Operators used in expressions are functions, while the various kinds of assignment are activities.
- A set of primitive statements is provided for repetitive and conditional computations. In particular, there is a uniform syntax for writing conditional statements and expressions (contrary to Verilog or VHDL, see figure 5)
- All objects used in a description must be declared. A TYPE defines a domain of objects, and operations on these objects. A CLASS is a set of types. Two classes are of particular interest: the VALUES and the CARRIERS (value holders).

An additional set of primitives is restricted to the derivation of languages from existing ones, with inheritance, extension and hiding mechanisms.

- The definition of a new TYPE or CLASS may be parameterized with types and classes. It is defined from an existing type, and may carry one, several, or all operations from the parent type. It may also define new operations, and extend the syntax to call these operations with infix operators. Operators may be overloaded.

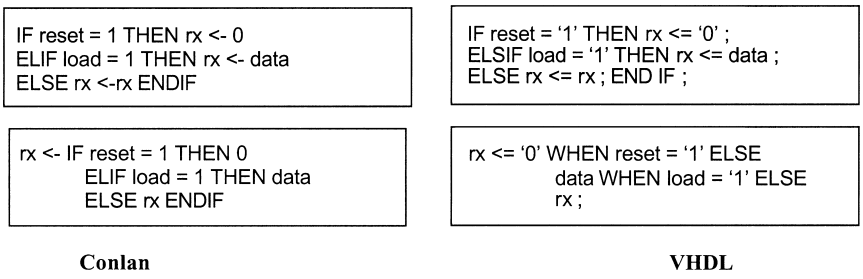


Figure 5: Conditional statement (top) and expression (bottom)

- The language designer may define new grammar rules, or assign new meanings to existing ones.
- Additionally, to specify extensions to the dynamic semantics, operation definitions characterized as INTERPRETER@, are invoked at the end of each computation cycle over the model. This is specially useful to define the memory properties of new carrier types.

Each member language of the Conlan family inherits from BCL facilities to describe the structure and the behavior of hardware, and inherits all or selected data types and operations.

IV . 1. Formal language definition principles

In order to test the extension mechanisms, the Conlan working group defined Base Conlan formally, in terms of a more primitive level called Primitive Set Conlan (PSCL), using only the semantic and syntactic extension mechanisms discussed above. Figure 6 shows the type derivation of BCL from the primitive types of PSCL.

In PSCL, the only available types are:

- the set of all possible values, called the universe of objects: type u n i v @,
- the set of integers, with the usual comparison and arithmetic operators: type i n t ,
- the sequences of characters of arbitrary length, type s t r i n g ,
- the Booleans, with the usual comparison and logical operators: type b o o l ,
- the sequences of objects of any type (possibly mixed): type t u p l e @,
- the primitive container, parameterized by the type of its value, called the cell: type c e l l @.

In addition, PSCL contains one class: the set of all possible types, denoted any@.

Subtypes and extended types are defined from these primitive concepts (Figure 6), using the type, function and procedure definition mechanisms of Conlan. The main types of BCL are:

- The positive integers, the natural integers and the interval between two bounds are subtypes of i n t .

- The structured data types are formally defined from the notion of tuple: arrays together with their dimension and indexing mechanisms, and records together with their fields.
- With another set of definitions, the notion of computation step and time signals, which correspond to the sequence of values of a signal along computation cycles and real simulated time, are also defined from t u p l e @.
- Finally, from the primitive cell, the basic generic carrier types are derived: t e r m i n a l (representing wires), v a r i a b l e (equivalent to the VHDL signal), and r t - v a r i a b l e (modeling the elementary master-slave flip-flop). Their definitions include the specification of their memory properties.

Figure 7 illustrates a simple type definition. The domain of type o c t a l is a subset of int defined by its characteristic property: it is the set of i n t elements between 0 and 7. The comparison functions e q u a l , n o t e q u a l are carried from type i n t , but no other arithmetic operator coming from i n t is made available. Function p l u s is redefined. Its result is computed in terms of the + and MOD operators from i n t : type cast between the parent type and the type being defined is specified using the conversion functions n e w and o l d . Statement FORMAT@ introduces syntactic and semantic extensions. In this

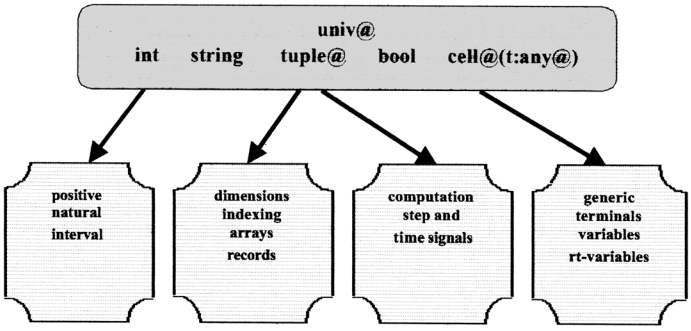


Figure 6: BCL type derivations

```
TYPE octal BODY
    ALL a:int WITH (a >= 0) & (a < 8) ENDALL
    CARRY equal, notequal ENDCARRY
    FUNCTION plus (x, y : octal) : octal
        RETURN new ((old (x) + old (y)) MOD 8)
        FORMAT@ EXTEND expression-5.2
                       MEANS plus (@1, @2) ENDFORMAT
    ENDplus
ENDoctal
```

Figure 7: Example Type Definition

example, a new meaning is given to non-terminal e x p r e s s i o n - 5 . 2 (which, in the syntax, expands into an infix call to operator +); the effect is to overload + to also mean a call to function p l u s on o c t a l parameters.

This example shows that a new type is defined from a parent type, but need be neither a strict extension nor a restriction of the parent type.

IV . 2. Conlan model of computation

A Conlan description is based on a synchronous data flow model of computation. Time is divided into units, and all delays in the description are integer multiples of that unit. The interpretation algorithm maintains two global counters: a time counter t@ and a computation step counter s@. As shown on Figure 8, the values held in the carriers of a circuit model are two dimensional sequences of values, called signals in Conlan: each time interval is divided into a sequence of computation steps, of varying number from time interval to time interval. Computation steps are added until the model is stable, i.e. all carriers have identical values in the last two steps.

A computation step consists in executing all concurrent activities in a description. These activities come down to the (possibly conditional) assignment of the next step value of the carriers, as a function of the current step values and past time values of one or more arriers.

Next values <- F(Current values, Past values)

Conceptually, for each time interval, there are as many values in a signal as computation steps. In practice, for a time interval, only the previous step value of a signal is accessible to compute the current step value of this and other signals; step values anterior to the previous one are no longer accessible, and can be removed. Likewise, only the last step value, the stable one, characterizes the value of a signal at a past time interval. Thus, at the end of each computation step, all signals are shrunk, and only one step value is kept.

Conlan has the concept of a INTERPRETER@ FUNCTION or ACTIVITY. It is the definition, in a type, of an operation executed under the control of the simulator. In particular, each carrier type definition embeds an INTERPRETER@ ACTIVITY that defines, in algorithmic form, the actions taken at the end of a computation step, characterizing the temporal behavior of the carrier type. For instance, if not assigned during the step, the carrier may get as step value:

- its previous time interval value (memory over time intervals), type r t - v a r i a b l e
- its previous step value (memory from step to step), type v a r i a b l e
- or a default value (no memory), type t e r m i n a l .

Thus, BCL is given operational semantics, defined in the language by an abstract simulator.

At the most primitive level, elementary operational semantics of the basic PSCL statements are given by an underlying model of concurrent agents (which shall not be further discussed here).

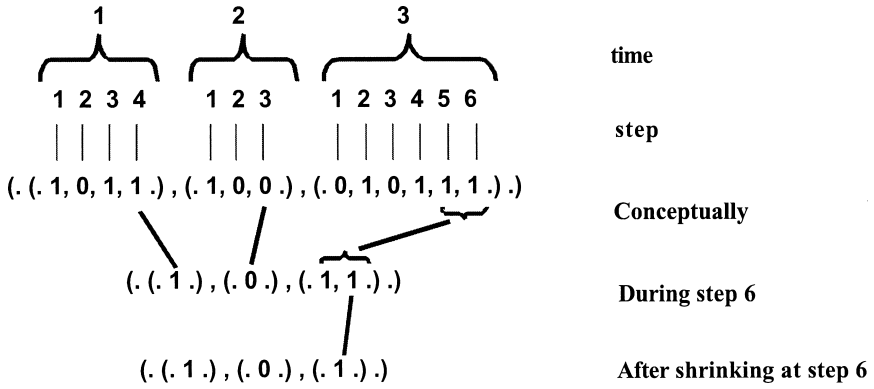


Figure 8: Signal: history of values

IV . 3. BCL as a hardware description language

BCL is not only the basic language layer from which all other languages of the Conlan family are derived, it is a proper HDL in itself. The reader will get a flavor of what writing in BCL looks like, on a famous example: the control algorithm for a drink dispenser machine. A drink costs 35 cents of the U.S.A. dollar. We assume that the machine has a single coin slot, and that the coin recognition mechanism and clock cycle are extremely fast compared to the coin inter-arrival time, so that at most one coin is present and not yet processed. If more than 35 cents has been received, the machine gives some change back, one coin of each kind (dime, nickel) at a time.

Figure 9a shows the interface of the circuit. Each of the three Boolean inputs is set to 1 when a corresponding coin has arrived; at most one of them is 1. Two Boolean outputs correspond to the return of change, and the third output commands the drink delivery.

Figure 9b gives the state transition diagram of the circuit, modeled as a finite state machine (FSM). Starting from the initial state *idle*, each coin leads to the state that tells the accumulation of money received so far. The arrows are labeled with the input that caused the state transition, possibly followed by the outputs positioned as a result, if any ("/" separates inputs and outputs, "D" abbreviates *DRIN*K). By default, no transition is taken.

Figure 9c gives excerpts of the behavioral description of figure 9b, written in BCL (for reasons of space, some of the states, which are very similar to the ones fully spelled out, are omitted). REFLAN announces the reference language used, here bcl. DESCRIPTION drinkmachine is followed by the declaration of the interface elements, here of type Boolean rt_variable with default value 0. The use of rt_variable for interface and the internal carrier fsm_state ensures that the model state changes at most once per time unit. TYPE state_name defines mnemonics for the FSM states as an enumerated data type, and carries all operators, namely equal and not equal, from the parent type, which in this case is univ@.

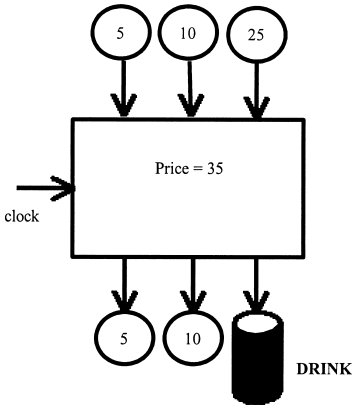


Figure 9a: Circuit interface

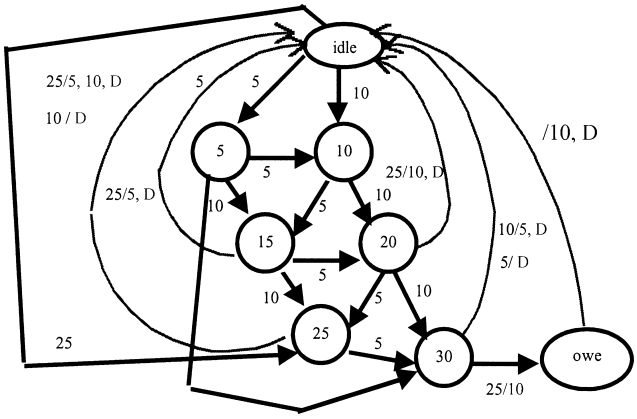


Figure 9b: Drink Dispenser State Diagram

The body of the description is synchronized by the rising edge of the clock, which is expressed by the enclosing IF statement. The condition ~clk%1 & clk reads: "not clk delayed one unit of time and clk", and therefore means "rising edge of clk". The CASE statement enclosed in the IF describes the control automaton, fsm_state holds the current state. For each possible value of fsm_state, the next value of fsm_state and of the outputs is computed as a function of the inputs. All the assignments executed for any alternative value of fsm_state are concurrent (they are separated by commas).

```
REFLAN bcl
DESCRIPTION drinkmachine
    (  IN clk, reset, nickel, dime, quarter : rt_variable(bool, 0);
       OUT nickel_out, dime_out, drink: rt_variable(bool, 0)  )
BODY
    TYPE state_name BODY
       { 'idle', 'five, 'ten', 'fifteen', 'twenty', twenty_5', 'thirty', 'owe_dime' }
       CARRYALL
    ENDstate_name
    DECLARE fsm_state : rt_variable(state_name , 'idle') ENDDECLARE
    IF ~clk%1 & clk THEN
       CASE state IS
       'idle':     nickel_out <- 0 , dime_out <- 0 , drink <- 0 ,
                   fsm_state <- IF nickel = 1 THEN 'five'
                        ELIF dime = 1 THEN 'ten'
                        ELIF quarter = 1 THEN 'twenty_5'
                        ELSE 'idle' ENDIF;
       'five':     ...
       'ten' :     nickel_out <- 0 , dime_out <- 0 , drink <- quarter,
                   fsm_state <- IF nickel = 1 THEN 'fifteen'
                        ELIF dime = 1 THEN 'twenty'
                        ELIF quarter = 1 THEN 'idle'
                        ELSE 'ten' ENDIF;
       'fifteen':  ...
       'twenty':  ...
       twenty_5':  ...
       'thirty':   nickel_out <- dime | quarter, dime_out <- quarter ,
                   drink <- nickel  | dime | quarter,
                   fsm_state <- IF nickel  | dime  THEN 'idle'
                        ELIF quarter = 1 THEN 'owe_dime'
                        ELSE 'thirty' ENDIF;

       'owe_dime':  nickel_out <- 0 , dime_out <- 1 , drink <- 0 ,
                   fsm_state <- 'idle';
      ENDCASE
    ENDIF
ENDdrinkmachine
```

Figure 9c: Drink Dispenser behavioral description in BCL

IV . 4. Conlan implementation

Two projects have implemented the concepts of Conlan (see Figure 10).

The EEC-funded multinational CASCADE project was one of the very first attempts to integrate, in a single language framework, the analog and discrete simulation paradigms [BH83, Me83]. The language definition, the compiler and mixed mode simulator implementation were performed at the University of Grenoble (France), in cooperation with Politecnico di Torino (Italy), under the coordination of Jean Mermet. Three discrete description levels and one constraint specification language were formally derived from BCL:

- CASSANDRE, the logic and bit-vector RTL
- LASCAR, the arithmetic and microprogram level

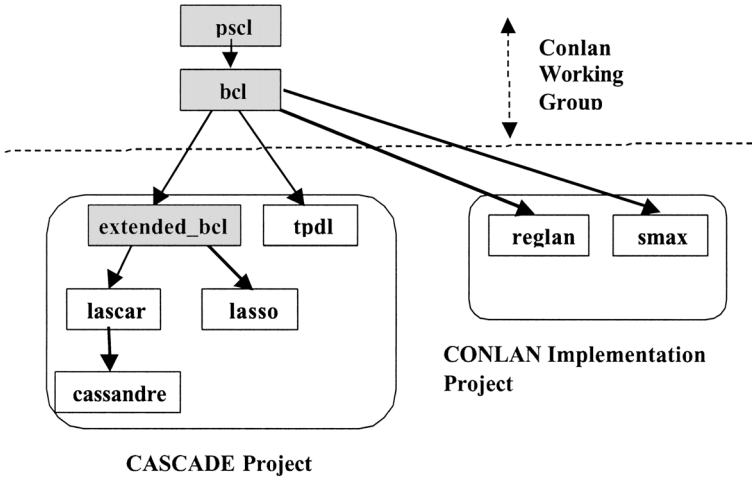


Figure 10: Circuit interface

- LASSO, the behavioral level
- TPDL, a "temporal profile description language" for the description of temporal properties [CCC85]

The CONLAN Implementation project, carried out at the Technical University of Darmstadt under the direction of Robert Piloty, has been the only effort to realize the *full* BCL-based description and language definition mechanisms. The project resulted into:

– REGLAN, a HDL formally derived from BCL [MS89, Pi93]
– a compiler front-end generator, including all the language and syntax extension primitives [DLM89]
– IREEN: a language-independent intermediate format [PW88]
– a back-end compiler
– a discrete simulator
– a design data-base system

The REGLAN language and simulation system have been used in Darmstadt to support practical classes on digital systems design. The language extension capabilities have also been used in performing research on HDL-based formal verification, both in Darmstadt where the SMAX language was defined for logic-level reasoning[Ev86], and in Marseille where theorem proving from LASCAR was investigated.

IV . 5. The influence of Conlan on VHDL

The essential principles of Conlan were already published when the VHDL project was started, and they were presented at the IDA workshop that defined the requirements for

VHDL, in the summer of 1981. The syntactic flavor of the two languages is very different, Conlan being Pascal-like, while VHDL is based on ADA. VHDL has benefited from significant advances brought by the ADA language, among which the most innovative in the domain of HDL's are:

- the separation between interface and body in descriptions and packages, and the associated concept of configuration (the constitution of a component as an entity-architecture pair);
- the existence of sequential statements to write algorithmic processes.

Yet, the influence of Conlan is manifest in four semantic features[BPH92, BP93]:

- The two-level model of time, that clearly distinguishes time and computation cycle, is one of the main innovative concepts of Conlan. It has been taken over in VHDL.
- The notion of generic parameter to an entity, that allows to define a family of closely related models, and the static binding of generic parameters at the point of component instantiation, come from generic descriptions in Conlan.
- The type definition of VHDL is a restricted version of the Conlan one. VHDL has a more simple function and operator overloading capability, and no true syntax extension. However, the subtype definition is identical.
- Conlan was the first HDL to introduce an assertion statement, associated either to the interface of descriptions and/or function parameters to check their proper usage by the enclosing environment, or concurrently evaluated with the description statements to observe the design behavior. VHDL took over this idea, adding the ability to issue a user-defined message upon assertion violation.

V. Conclusion

Looking back at the history of Hardware Description Languages, Robert Piloty played a very significant role in the elaboration of concepts and in the structuring of the community of researchers in this area, at the international level.

The most renowned scientific event has been the Conference on "Computer Hardware Description Languages and their applications", called "CHDL"[CHDL74-93]. The first four venues were organized under the auspices of IEEE and ACM: New Brunswick (New Jersey, USA, 1973), Darmstadt (Germany, 1974), New York City (USA, 1975), Palo Alto (California, USA, 1979). It is worth noticing that Robert Piloty organized and chaired the second conference, the only one of the four which took place outside of the USA.

Extremely active in the International Federation for Information Processing (IFIP), Robert Piloty was a founding member and the German representative in the IFIP Technical Committee number 10 on Computer Hardware in 1976 (now renamed Computer Systems Technology); in 1980, he formed the working group 10.2 on "Digital Hardware Description and Design Tools", of which he was the first chairman. After its creation, IFIP WG10.2 sponsored and organized the following "CHDL" conferences as a bi-annual event, successively in Kaiserslautern (Germany, 1981), Pittsburgh (USA, 1983), Tokyo (Japan, 1985), Amsterdam (The Netherlands, 1987), Washington D.C. (USA, 1989), Marseille

(France, 1991), Ottawa (Canada, 1993), Chiba (Japan, 1995), Toledo (Spain, 1997). What should have been the 14 th venue in 1999 was cancelled for lack of submitted contributions: HDL's had long ceased to be a research topic, and the "CHDL" Conference was more and more dedicated to design tools and formal methods, which its title did not show. In parallel, a more application oriented "Asia-Pacific CHDL" Conference had been formed under the auspices of the same IFIP WG10.2, with meetings in Brisbane (Australia,1993), Toyobashi (Japan, 1994), Bangalore (India, 1996), Taipeh (Taiwan, 1997), Seoul (Korea, 1998), Beijing (China, 2000). In Europe, the "Forum on Design Languages" has been formed in 1998 as a yearly event in September, focusing on standardization efforts, user experience and tools; meetings were held in Lausanne (Switzerland), Lyon (France) and Tübingen (Germany).

With the standardization of VHDL and Verilog, the availability of efficient synthesis software starting from register transfer level descriptions, and the need for fast simulation, the interest of the scientists shifted away from Hardware Description Languages. HDL's can be considered a mature technology. Hot topics in research include all technological aids to face the challenges of an ever increasing design speed, and short life time of the product. A drastic reduction in design time requires to automate more tasks at higher levels. The real challenges of today are named verification, built-in fabrication test, fault immunity, reduction of power consumption, ...

The design of processor-like numeric circuits is now concentrated in few industrial sites. Data processing is no longer the main issue, but rather system on a chip is getting more attention. The consequence is a focus on the heterogeneous aspects of a system: mixing hardware and software, analog and digital, batteries, micro-electro-mechanical sensors and actuators. Given the variety of the needed modeling paradigms, no single language can meaningfully encompass them all while remaining tractable. System-level specification and design must be expressed in terms of the communication and the synchronization between heterogeneous sub-systems, each described in the most appropriate formalism. What is missing is a sound semantic definition for such communication.

Strangely enough, the "system-on-a-chip" design teams are faced again with a new "Tower of Babel" but this time at system level. Multi-language simulation systems are weak in the semantic definition of inter-language communications, and do not seriously guarantee that what you implement will perform as what you simulate. Many system-level formalisms are proposed: UML, Rosetta, Esterel, SDL, CSP … the list is far from exhaustive. Back to the situation of the mid-seventies, we need a new CONsensus System Description effort, to identify and formally define the basics of communication between semantic paradigms, and provide sound construction mechanisms for complex notions from simple ones.

VI. References

[BKS74] M. Becker, R. Klar & P.P. Spies: The Erlangen Computer Design Language, in [CHDL74], pp.176-185

[BHL83] D. Borrione, M. Humbert & C. Le Faou : Hierarchical Mixed-mode simulation in the CASCADE project, Proc. VHDL Conference, Trondheim, 1983

[BL87] D. BORRIONE & C. LE FAOU: Implementation Techniques for Multi-Level Hardware Description Languages, in Hardware Description Languages, ed. R.W. Hartenstein, North-Holland, 1987, pp. 409-437.

[Bo75] D. BORRIONE: LASCAR: A Language for Simulation of Computer Architecture, in [CHDL75], pp.143-152.

[BN71] C.G. BELL & A. NEWELL: Computer Structures: Readings and Examples, ed. McGraw Hill Book Company, New York, 1971.

[BP93] D. BORRIONE & R. PILOTY: CONLAN: Presentation of Basic Principles, Applications and Relation to VHDL, in Fundamentals and Standards in Hardware Description Languages, ed. J. Mermet, NATO ASI Series, Vol. 249, 1993, pp. 39-78.

[BPH92] D. BORRIONE, R. PILOTY, D. HILL, K.J. LIEBERHERR & P. MOORBY: Three Decades of HDLs: Part 2, Conlan Through Verilog, in IEEE Design & Test of Computers, Field-Programmable Gate Arrays, September 1992, pp. 54-63.

[CCC85] G. CABODI, P. CAMURATI, G. CROSSETTI &P. PRINETTO: Experiences in CONLAN based formal verification of HDL's, in [CHDL85]

[CDD92] Y. CHU, D.L. DIETMEYER, J.R. DULEY, F. J. HILL, M. R. BARBACCI, C.W. ROSE, G. ORDY, B. JOHNSON & M. ROBERTS: Three Decades of HDLs: Part 1, CDL Through TI-HDL, in IEEE Design & Test of Computers, VHDL Development, Prescription for Success, June 1992, pp. 69-81.

[CHDL74] Proceedings of Workshop on Computer Hardware Description Languages, ed. R. Piloty, Darmstadt University, Germany, July 31 – August 1 and 2, 1974, pp. 1-240.

[CHDL75] Proceedings of International Symposium on CHDL and Their Applications, Graduate Center, City University of New York, New York City, USA, September 3-5, 1975, ed. S.Y.H. Su and D.L. Dietmeyer, pp. 1-191.

[CHDL79] Proceedings of 4th International Symposium on CHDL, Palo Alto, California, USA, October 8-9, 1979, ed. W.M. vanGleemput and D. Dietmeyer, pp. 1-190.

[CHDL81] Proceedings of the 5th International Conference on CHDL and Their Applications, Kaiserslautern, Germany, 7-9 September, 1981, ed. M. Breuer and R. Hartenstein, North-Holland, pp. 1-349.

[CHDL83] Proceedings of 6th International Symposium on Computer Hardware Description Languages and their Applications, Carnegie-Mellon Univ., Pittsburgh, Pennsylvania, USA, May 23-25,1983, ed. T. Uehara and M. Barbacci, North-Holland, pp. 1-243.

[CHDL85] Proceedings of 7th International Symposium on Computer, Tokyo, Japan, Aug.29-31, 1985, ed. C.J. Koomen and T. Motooka, North-Holland, pp. 1-493A

[CHDL87] Proceedings of the 8th International Symposium on CHDL and Their Applications, Amsterdam, The Netherlands, 27-29 April, 1987, ed. M.R. Barbacci and C.J. Koomen, North-Holland, pp. 1-405.

[CHDL89] Proceedings of 9th IFIP Symposium on CHDL and Their Applications, June 19-21, 1989, Washington, DC, USA, ed. J.A. Darringer and F.J. Rammig, pp. 1-361.

[CHDL91] Proceedings of 10th International Symposium on CHDL and Their Applications, Marseille, France, 22-24 April, 1991, ed. D. Borrione and R. Waxman, North-Holland, pp. 1-477.

[CHDL93] Proceedings of the Conference on CHDL and Their Applications, Ottawa, Canada, 26-28 April, 1993, ed. D. Agnew, L. Claesen and R. Camposano, pp. 1- 590.

[DLM89] V. Dedecke-Beuttner, M.N. Lipp, R.T. Menche, Th. Michl & A.S. Schmitt: REGLAN: User Manual (V1.20), October, 1989, Technische Hochschule Darmstadt, pp. 1-72.

[Ev86] H. Eveking: SMAX – A CONLAN member language for verifiable hardware descriptions. Proc. Euromicro'86, North Holland, pp. 549-558

[FMS75] P.L. Flake, G. Musgrave & M. Shortland: The HILO Logic Simulation Language, in [CHDL75], pp. 134-142.

[Ha93] R.W. Hartenstein: KARL and ABL, in Fundamentals and Standards in Hardware Description Languages, ed. J. Mermet, NATO ASI Series, Vol. 249, 1993, pp.447-466.

[Ie87] Standard VHDL Language Reference Manual, IEEE Std 1076-1987.

[Ie93] Standard VHDL Language Reference Manual, IEEE Std 1076-1993.

[Ie95] 1364-1995 IEEE Standard Description Language Based on the Verilog© Hardware Description Language, 1995.

[Ie99] Standard for VHDL Register Transfer Level (RTL) Synthesis, IEEE Std 1076.6-1999

[Li77] G.J. Lipovski:Hardware Description Languages: Voices from the Tower of Babel", in Computer, Vol. 10, No. 6, June 1977, pp. 14-17.

[ME73] Etude Méthodologique de la Conception Assistée par Ordinateur des Systèmes Logiques : CASSANDRE, J. Mermet, thèse d'Etat, Université Scientifique et Médicale de Grenoble, 10 avril 1973.

[ME83] J. Mermet: Circuit And System Computer Aided Design and Engineering, in Proceedings of CAPE'83:1 st International Conference on Computer Applications in Production and Engineering, Amsterdam, The Netherlands, April 1983, pp. 245-262.

[MN93] J.D. Morrison & C.O. Newton: ELLA, a Language for the Design of Digital Systems", in Fundamentals and Standards in Hardware Description Languages, ed. J. Mermet, NATO ASI Series, Vol. 249, 1993, pp. 385-394.

[MMR98] J. Mermet, P. Marwedel, F.J. Rammig, C. Newton, D. Borrione & C. Le Faou: Three Decades of Hardware Description Languages in Europe", in Journal of

Electrical Engineering and Information Science, Vol. 3, No. 6, December 1998, pp. 700-721.

[MS89] R.T. MENCHE & A.S. SCHMITT, REGLAN: Language Reference Manual (V1.20), September, 1989, Technische Hochschule Darmstadt, pp. 1-167.

[PBB83] R. PILOTY, M. BARBACCI, D. BORRIONE, D. DIETMEYER, F. HILL & P. SKELLY: CONLAN Report, Lecture Notes in Computer Science, No. 151, Springer Verlag, 1983.

[Pi69] R. PILOTY:RTS I, Research Report, Nachrichtenverarbeitung, TH Darmstadt, 1969

[Pi75] R. PILOTY: Segmentation Constructs for RTS III, A Computer Hardware Description Language Based on CDL", in [CHDL75], pp. 115-124.

[Pi93] R. PILOTY: REGLAN, in Fundamentals and Standards in Hardware Description Languages, ed. J. Mermet, NATO ASI Series, Vol. 249, 1993, pp. 431-446.

[PW88] R. PILOTY & B. WEBER: IREEN – A Datamodel for Toolintegration in Open Microelectronic CAD – Systems, in Proceedings of Workshop on Tool Integration and Design Environments, Paderborn, Germany, 26-27 November, 1987.

[Ra75] F.J. RAMMIG: DIGITEST II: An Intergrated Structural and Behavioral Language, in [CHDL75], ed. IEEE Computer Society, pp. 38-44.

[Ra93] F.J. RAMMING : The Hardware Description Language DACAPO III, in Fundamentals and Standards in Hardware Description Languages, ed. J. Mermet, NATO ASI Series, Vol. 249, 1993, pp.395-409.

Prof. Dr. Dominique Borrione · Laboratoire TIMA
46 Avenue Felix Viallet · 38031 Grenoble cedex · France